Lars Roith

(E-Mail: lars.roith@aitag.com)
ist für die AIT als Berater im AIT Team SystemPro Team tätig. Zu seinen Schwerpunkten gehört neben der Projektleitung das Application-Lifecylce-Management. Er berät Unternehmen bei der Einführung und Anpassung des Visual Studio Team Foundation Servers. Seine Erfahrungen vermittelt er zudem als Autor des TFS-Blogs in Magazinen und in Vorträgen.

Distributed Scrum: Agiles Arbeiten in verteilten Teams – ein Erfahrungsbericht

Eine Software zu entwickeln, die über 700 einzelne Anforderungen enthält und innerhalb von 3 Jahren marktreif sein muss, ist eine große Herausforderung. Dies mit einem verteilten Team zu realisieren, das in diesen 3 Jahren auch mehr als 220 Anforderungsänderungen integriert und mehr als 1000 gemeldete Fehler behebt, scheint ein waghalsiges Unterfangen. Wir haben zusammen mit unserem Kunden und unseren Partnern diese Herausforderung angenommen. Das Ergebnis sind über zwei Millionen Zeilen Quellcode, die im kompilierten Zustand eine Windowsapplikation zur Definition, Durchführung und Auswertung von Messungen bilden. Wir haben das Projekt ohne das Bewusstsein, agil Arbeiten zu wollen, gestartet. Die Applikation und der Fortschritt an Funktionalität standen stets im Fokus und rückblickend lässt sich sagen: Das, was wir gemacht haben, bezeichnen viele als Scrum oder Agil. Nun, was haben wir getan? Wie konnten wir trotz örtlicher Trennung so flexibel arbeiten? Was war notwendig, um trotz der Verteilung der Teams erfolgreich eine funktionstüchtige Software zu liefern? Im Folgenden sollen die Projektphasen, unsere Vorgehensweisen, die Herausforderungen und unsere Lösungen vorgestellt werden, um zu zeigen, wie in verteilten Teams agil gearbeitet werden kann und Scrum trotz eines verteilten Teams zum Erfolg wird.

Projektstart – Teil 1

Das Projekt startete im eigentlichen Sinne zweimal. Der erste Start war der Beginn der Anforderungssammlung. Zu diesem Zeitpunkt ähnelte das Vorgehen eher dem Wasserfall-Modell. Um eine Auftragsvergabe seitens des Kunden durchführen zu können, war es notwendig, zu wissen, was am Ende als Ergebnis herauskommen soll. Daher wurde in enger Zusammenarbeit ein Anforderungsdokument erstellt, aus dem die angestrebte Funktionalität hinreichend hervorging. Auf weniger als 150 Seiten in Microsoft Word wurde ein System beschrieben, welches am Ende von maximal drei Jahren Entwicklungszeit an Endkunden ausgeliefert werden sollte. Zur Aufwandsabschätzung wurden zwei verschiedene Methoden eingesetzt. Zum einen wurden vier parallele Abschätzungen durch unterschiedliche Personen durchgeführt.

Die unterschiedliche Risikobewertung von neuer Technologie seitens des Kunden und unbekannter Problemdomäne seitens der Entwicklung führte zu großen Abweichungen der Abschätzungen. Zum Ausgleich des Risikos wurde eine zweite, stark vereinfachte Abschätzung auf Basis von Anforderungsbereichen und Komplexitäten vorgenommen. Dabei wurde die Umsetzung des Data Layers abgeschätzt, die Aufwände zur Erstellung des Business Layers und der Benutzeroberfläche wurden anhand von Faktoren bestimmt. Das Grundmaß für die Abschätzung war dabei „Ideal Days“, eine aus dem Extreme Programming bekannte Größe zur Aufwandsabschätzung. Das Ergebnis dieser Abschätzung lag sehr nah am Mittelwert der Best Case-Abschätzungen. Dies festigte den Projektrahmen: In 3 Jahren Laufzeit müssen ca. 20 Personenjahre Entwicklungsleistung erbracht werden. Diese Festlegung stellte, nachdem von allen Beteiligten die Zustimmung vorlag, in der genannten Zeit ein positives Ergebnis erreichen zu können, den Startschuss für das eigentliche Projekt dar.

Projektstart – Teil 2

Nachdem im ersten Schritt die grundlegenden projektorganisatorischen Punkte sowie ein klares Bild von der Funktionalität und dem Rahmen der Anwendung geklärt waren, wurde die Teamorganisation in Angriff genommen. Hierbei wurden zwei wesentliche Punkte festgelegt: Verantwortlichkeiten und grundsätzliche Kommunikation.

TIPP: Unter dem Gesichtspunkt „Verantwortlichkeit“ wurden folgende, für das Projekt wesentliche Festlegungen getroffen:

- *Das Team soll in der Lage sein, schnell produktrelevante Entscheidungen zu treffen. Um dies zu gewährleisten, wurde der Produktmanager als vollwertiges Teammitglied integriert. Er nahm an allen Besprechungen teil und konnte jederzeit seine Anforderungen konkretisieren.*
- *Das Team soll in der Lage sein, schnell Entscheidungen zur Teamstruktur zu*

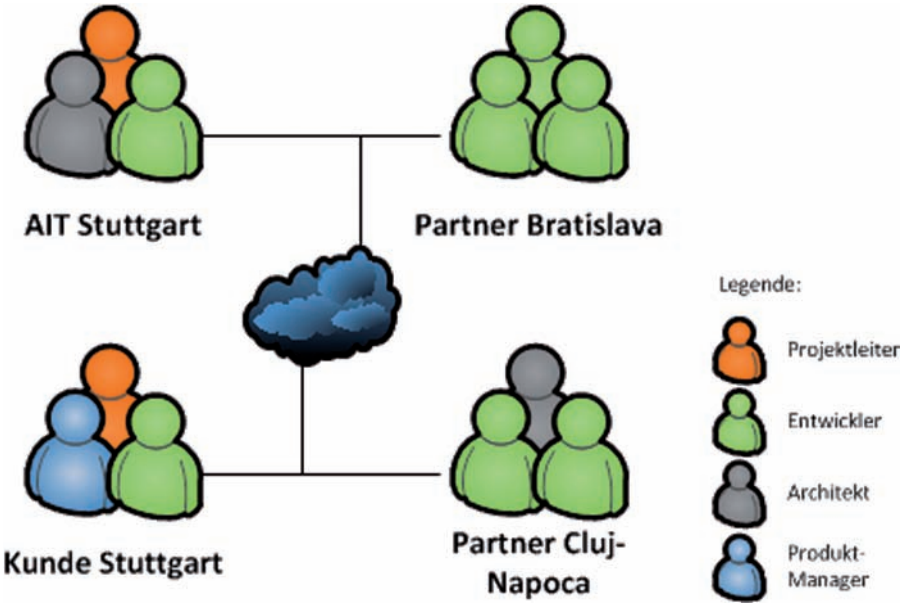

Abb. 1: Funktionale und örtliche Verteilung des Teams

treffen. Hierfür übernahm der Projektleiter der AIT die Gesamtverantwortung für das Team inkl. der Mitarbeiter des Partners und des Kunden. Er hatte jederzeit die Möglichkeit, Teammitglieder infrage zu stellen oder gar auszuschließen sowie neue Mitarbeiter einzubinden. Diese Entscheidungsgewalt ermöglichte es, sehr schnell auf Engpässe reagieren zu können und zum richtigen Zeitpunkt das Team mit der passenden Ressource zu stärken.

- *50%-Regel: Jedes Teammitglied trägt die Verantwortung, innerhalb der ersten 50% der abgeschätzten Zeit einer Aufgabe sicherzustellen, dass die Aufgabe in den verbleibenden 50% fertiggestellt werden kann. Diese Festlegung führte dazu, dass Risiken stets als erstes adressiert wurden und somit jederzeit rechtzeitig reagiert werden konnte.*

Auch für die Kommunikation im Team wurden Regeln aufgestellt. Durch die örtliche Verteilung des Teams musste ein virtueller Team-Room geschaffen werden. Die Nutzung eines Instant Messengers – im vorliegenden Fall Skype [Sky] – lieferte hierfür die Lösung und brachte einen zweiten Vorteil mit sich: die Kommunikation wurde gleichzeitig protokolliert, was zu mehr Sicherheit bei Abstimmungen führte.

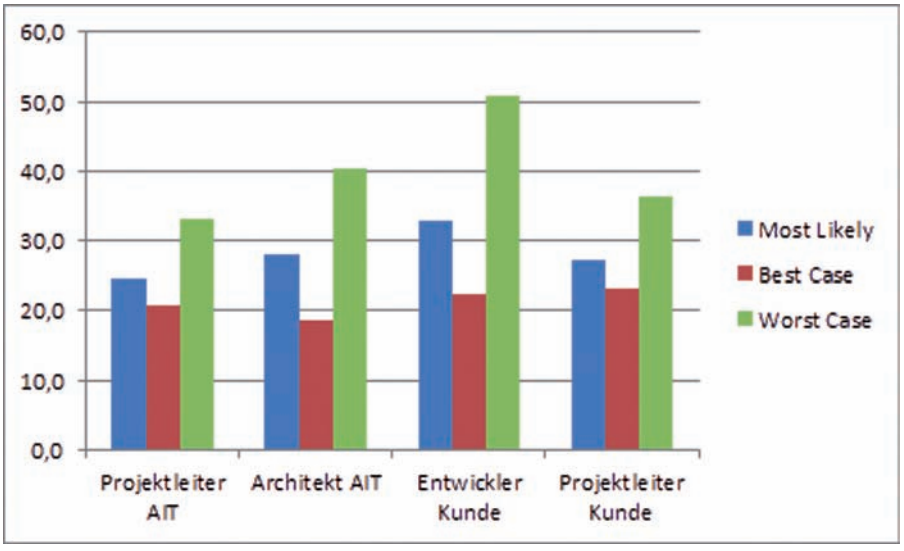

Abb. 2: Abschätzung des Projekts (in Personenjahren)

Mit dem Einsatz eines Instant Messengers traten aber auch Hürden auf, die genommen werden mussten. So verleitete die Nutzung von Skype zu einem „Nachrichten-Ping-Pong“, welches durch ein einfaches, kurzes Gespräch hätte vermieden werden können.

TIPP: Der Onlinestatus des Instant Messengers wird genutzt, um die Verfügbarkeit eines Teammitglieds anzuzeigen. Der Chat wird genutzt, um die Aufmerksamkeit des Kollegen zu erlangen. Bei Chats länger als 3 Nachrichten wird zum Telefon gewechselt.

Gleichzeitig musste ein Weg gefunden werden, wie verteilte Teammitglieder zugleich an einer Aufgabe arbeiten können. Aus diesem Grund wurden Desktop-Sharing-Applikationen – im vorliegenden Fall NetViewer [Net] – eingesetzt. Durch diese zwei Maßnahmen – Instant Messaging und Desktop Sharing – für die Kommunikation, konnte das Risiko der örtlichen Trennung nahezu vollständig eliminiert werden.

Ein weiterer, wichtiger Schritt für die reibungslose Zusammenarbeit in verteilten Teams war die Bereitstellung aller projektrelevanten Informationen in einem Repository – dem Microsoft Team Foundation Server (TFS) [MTFS]. Die in der Analysephase gesammelten und in Word dokumentierten Anforderungen wurden mithilfe von WordToTFS [AITW] als „Requirement“ Work Items in den TFS übernommen und dort mithilfe der AIT VSTS Extension [AITV] für die weitere Bearbeitung strukturiert und geplant (siehe Abbildung 3).

TIPP: Alle im Projekt entstehenden Artefakte, alle Aufgaben sowie aller Quellcode und alle Ergebnisse sollten in einem zentralen, für alle Teammitglieder zugänglichen System gehalten werden.

Projektplanung

Das Projekt hatte zur Bedingung, in Abständen von jeweils sechs Monaten, bestimmte, bei Projektvergabe definierte Meilensteine zu erreichen. Durch diese Meilensteine wurde der Umfang der einzelnen Releases festgelegt und eine Möglichkeit geschaffen, prüfen zu können, ob das geplante Gesamtziel rechtzeitig erreicht werden kann. Allerdings sind für die Durchführung sechs Monate ein zu großer Zeitraum für Planung und mögliche Kor-

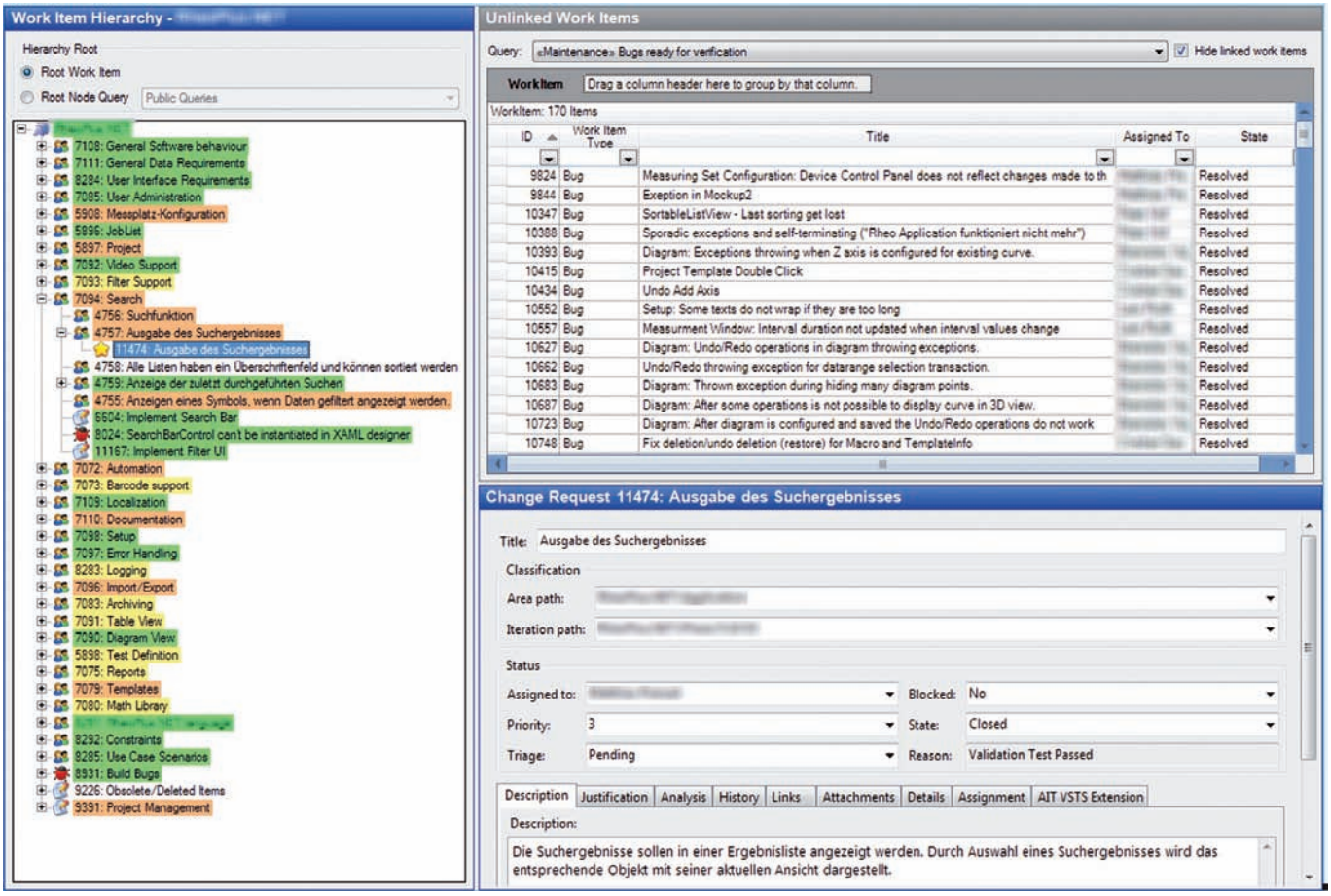

Abb. 3: AIT VSTS Extension zur Verwaltung von Anforderungen, Änderungen, Aufgaben und Fehlern

rekturen und ein viel zu langer Zeitraum für einen Sprint angesichts der erwähnten Projektrisiken.

Daher wurden die vorgegebenen 6-Monats-Zyklen unterteilt in jeweils sechs Unterzyklen, wodurch eine Iterations- oder Sprintlänge von einem Monat entstand. Jeder so entstandenen Iteration wurden die in dieser Iteration zu liefernden Anforderungen unter Zuhilfenahme des TFS zugeordnet. Die Planung der einzelnen Entwicklungsaufgaben wurde auf einer noch kleineren Einheit – der Woche – durchgeführt.

***TIPP:** Die Sprintlänge muss so gewählt werden, dass relevante Änderungen zeitnah einfließen können. Wird keine Sprintlänge definiert, werden einfließende Änderungen meist nicht hinreichend geklärt und oft nachträglich erneut verändert.*

Durch dieses Vorgehen war sichergestellt, dass die in einer Iteration zu liefernden Anforderungen bekannt waren und dass unter Berücksichtigung der vereinbarten 50%-Regel nur die jeweils wichtigsten detailliert betrachtet wurden. Jeder aktuell detailliert betrachteten Anforderung wurden daraufhin durch den verantwortlichen

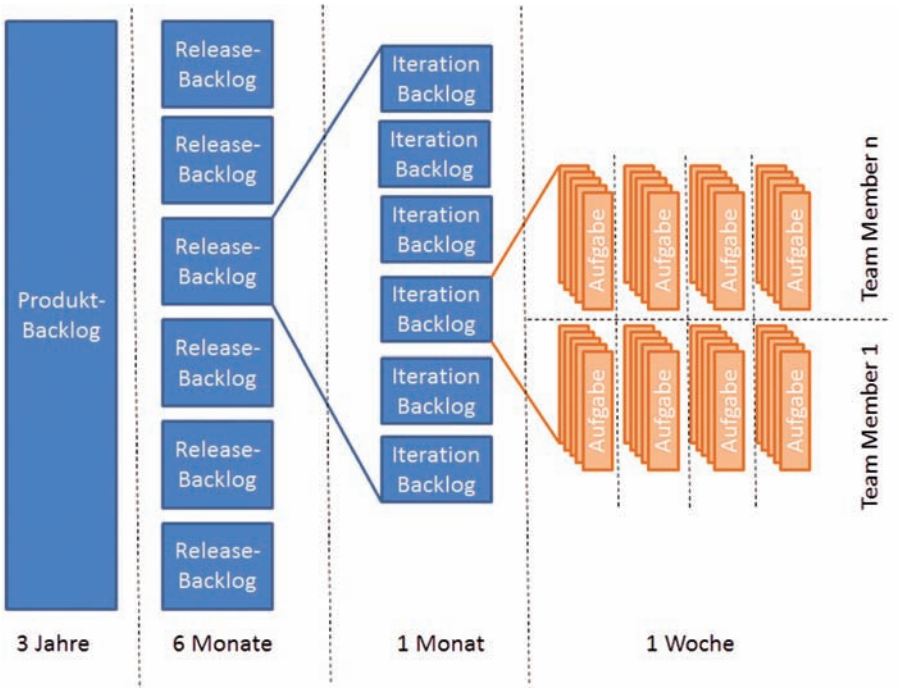

Abb. 4: Planungsgranularität

Entwickler die zur Umsetzung notwendigen Aufgaben zugeordnet und abgeschätzt.

Weiterhin wurden zwei feste Besprechungen etabliert. Ein „Daily Status Meeting“, welches, wie in Abbildung 4 gezeigt, in Form eines Gruppenchats gehalten wurde und meist nicht länger als 5-10 Minuten dauerte. Parallel dazu wurden in einem wöchentlichen, einstündigen Zusammentreffen größere Abhängigkeiten geklärt sowie das Feedback von Produktmanagern und den fachlichen Testern an das Team weitergeben. Diese Besprechung wurde auch genutzt, um Zwischenergebnisse im Team zu präsentieren, wodurch das Bewusstsein und das Verständnis für die Applikation bei jedem Entwickler verbessert wurden und Abhängigkeiten nicht nur durch das Produktmanagement, sondern auch durch die Entwickler selbst erkannt werden konnten.

Entwurf, Entwicklung und Test

Während die in der Anfangsphase des Projektes gesammelten und dokumentierten Anforderungen hinreichend für eine Aufwandsabschätzung waren, reichte ihr Detailgrad nicht aus, um sie direkt umsetzen zu können. Aus diesem Grund wurden zwei weitere Dokumente für die einzelnen Features erstellt.

Das erste Dokument lieferte Anwendungsfälle, welche in Form von visiobasierten Oberflächen-Prototypen und mittels Arbeitsschritten die Interaktion des Anwenders mit dem System beschrieben. Die gewählte Form der Dokumentation konnte zudem als Ausgangsbasis für die Testfälle genutzt werden. Durch die bereits beschriebenen Abläufe wurde der Aufwand für die Erstellung von Testfällen maßgeblich reduziert. Es mussten so nur noch die Testumgebungen sowie Akzeptanzkriterien definiert werden.

Als zweites Dokument wurde von jedem Entwickler die technische Dokumentation der entwickelten Lösung in Form eines Software-Design-Dokuments (SDS) geliefert. In diesem wurden der Bezug zu den umgesetzten Anforderungen verdeutlicht und die für die Lösung maßgeblichen Entscheidungen dokumentiert. Darüber hinaus wurden in diesem Dokument auch offene Punkte, Risiken und nicht adressierte Anforderungen festgehalten, sodass von jedem Mitarbeiter schnell nachvollzogen werden konnte, warum die Lösung in der vorliegenden Form entstand. Mit diesem Vorgehen entstanden während der

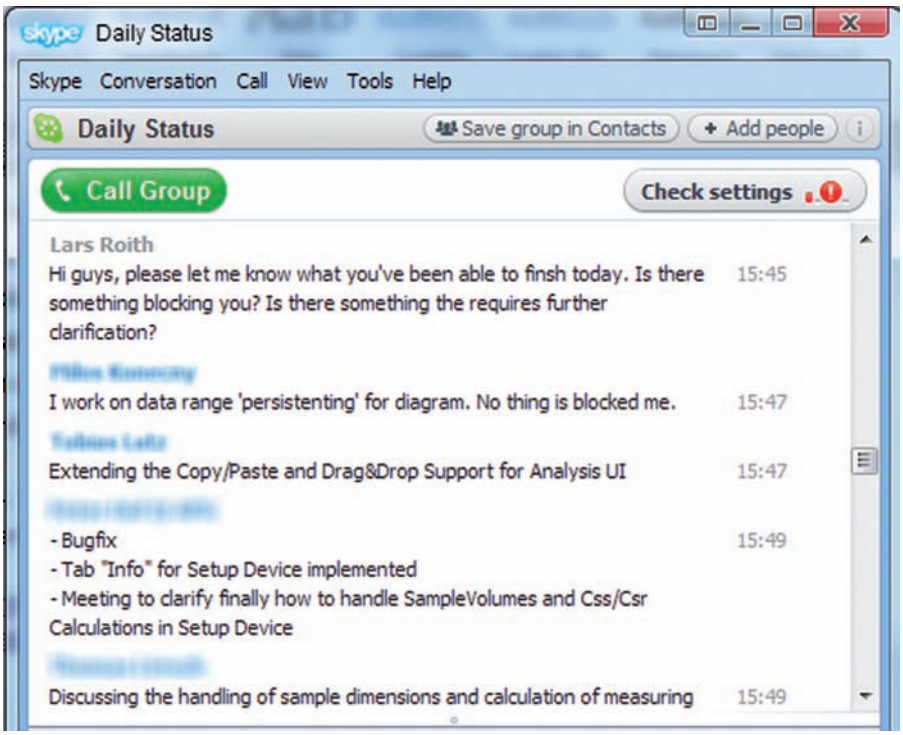

Abb. 5: Daily Status Chat

Projektdurchführung mehr als 800 Seiten Anwendungsfall- und somit auch Testfallbeschreibungen sowie mehr als 1500 Seiten Software-Design-Dokumentation.

Aus Abbildung 5 wird deutlich, dass die Entwicklung und der Test der Lösung nicht erst nach Beendigung der Anwendungsfallbeschreibung und der Software–Design-Spezifikation erfolgten, sondern bereits während der Erstellung dieser Dokumente aktiv bearbeitet wurden. Dieses Vorgehen führte zu Überlappungen, stellte jedoch sicher, dass kein Engpass in der Erstellung der Dokumente entstand. Gelegentlich bedingte dieses Vorgehen, dass Änderungen auftreten konnten, welche die Anpassung bereits entstandenen Quellcodes erforderten. Der Umgang mit diesen Änderungen erforderte zwei wesentliche Einstellungen im Team:

TIPP: Der Projektleiter muss darauf ver-

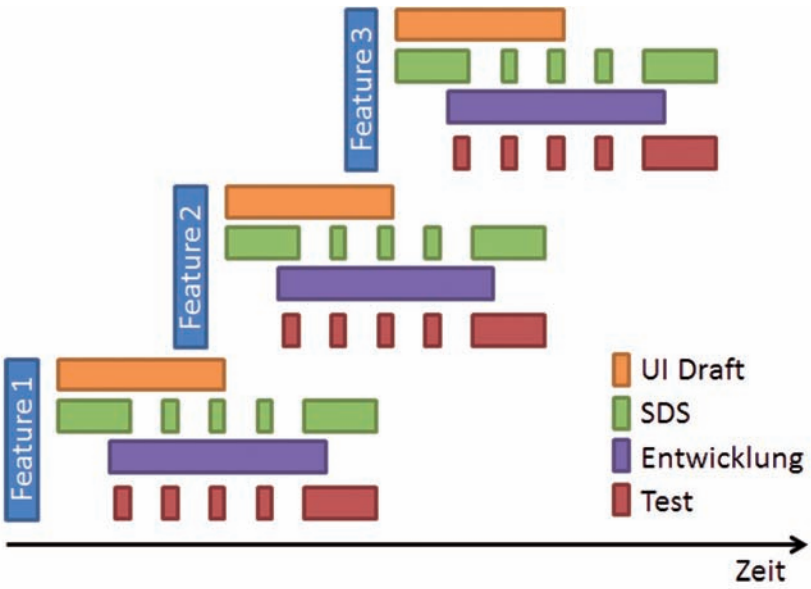

Abb. 6: Entstehung von Artefakten

Abb. 7: Projektportal zur Versionierung und Freigabe von Dokumenten

trauen können, dass der Architekt und die Entwickler die Software flexibel änderbar gestalten. Der Entwickler muss darauf vertrauen können, dass eine notwendig Anpassung aufgrund einer Änderung nicht als Entwicklungsfehler seinerseits betrachtet wird und dass ihm der für die Änderung notwendige Aufwand zugestanden wird.

Aufgrund der gestatteten Änderung von Anforderungen und deren Detailierung musste ein Weg gefunden werden, wie der für das System gültige Dokumentenstand ermittelt werden konnte. Hier half der TFS mit seinem Projektportal. Jedes Dokument wurde im Projektportal abgelegt und unterlag dort sowohl einer Versionierung als auch einer Freigabe. Dadurch war sichergestellt, dass z. B. Entwickler nur Oberflächen erstellten, die mit dem Produktmanagement abgestimmt und von diesem abgesegnet waren. Des Weiteren sind die Dokumente mit den zugrundeliegenden Anforderungs-Workitems im TFS verknüpft.

Vor der finalen Abgabe eines Features in das Versionskontrollsystem wurden die Dokumente durch das Produktmanagement sowie den Architekten abgenommen. Hinzu kamen manuelle Code-Reviews sowie automatisierte Tests für alle Komponenten. Bei der Testautomatisierung wurde ebenfalls auf die Funktionen des TFS zurückgegriffen und mit jedem Einspielen einer neuen Quellcode-Version ein Buildprozess angestoßen. Dieser Prozess hatte die Aufgabe neben dem eigentlichen Kompilieren der Software und dem Anstoßen der automatischen Tests auch die zugrundeliegenden Kodierrichtlinien und Softwaremetriken zu überwachen.

So konnte neben der kontinuierlichen Lieferung von Features auch laufend an der Verbesserung der Softwarequalität gearbeitet werden. Zudem sicherten die automatisierten Tests trotz auftretender Änderungen und Anpassungen die stete Lieferbarkeit einer lauffähigen, aktuellen Version.

Damit Fehler so zeitig wie möglich vermieden werden, wurde neben der Buildautomatisierung – welche ja immer erst nach dem Einspielen einer neuen Version in Aktion tritt – auch auf Regeln zurückgegriffen, die bereits das Einchecken von fehlerhaftem Quellcode verhindern. Darüber hinaus wurde jeder Eincheckvorgang mit der dazugehörigen Aufgabe verknüpft. Hierdurch wurde gewährleistet, dass bei der Analyse von geändertem Quellcode durch Nutzung der TFS-Funktionalität stets der Grund für die Änderung herausgefunden werden konnte.

TIPP: Alle Codeänderungen basieren auf abgenommenen Anforderungen, haben eine zugeordnete Aufgabe, werden einem Review unterzogen, automatisch integriert und hinsichtlich der Einhaltung von Richtlinien und Funktion getestet.

Fazit

Auch in verteilten Teams ist ein agiles Vorgehen möglich. Im Gegensatz zu örtlich verbundenen Teams müssen bei verteilten Teams aber wesentlich mehr Anstrengungen unternommen werden, um das Teamgefüge stabil, die Kommunikation flüssig und die Motivation aufrecht zu erhalten. Verteilte Teams benötigen dazu eine weitaus stärkere Identifikation mit dem gesamten Team, dem Produkt und dem Vorgehen.

Auch das gegenseitige Vertrauen im Team muss größer sein, gleichzeitig steigt die Verantwortung jedes Einzelnen. Unterstützt wird die Stabilität des Teamgefüges durch den Einsatz einer geeigneten Application Lifecycle Management-Plattform mit spezifischen Erweiterungen zur Überwindung der geschilderten Hürden. Maßgeblich für den Erfolg des Teams ist dabei aber nicht das Werkzeug an sich, sondern der korrekt dosierte und auf die Teamstärkung orientierte Einsatz des Werkzeugs. ■

Referenzen

[Sky] Skype: http://www.skype.com
[Net] Netviewer: http://www.netviewer.com/
[MTFS] Microsoft Team Foundation Server: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/team-foundation-server
[AITW] AIT WordToTFS: http://www.aitgmbh.de/word_to_tfs0.0.html
[AITV] AIT VSTS Extension: http://www.aitgmbh.de/vsts_extension_workitems.0.html